Teooria

1. *Implicit wait vs explicit wait. Millal kumb?*

Implicit wait seab üldise aja, kui kaua WebDriver otsib elementi enne veateadet - see kehtib kõikidele elementidele ja jääb aktiivseks kogu sessiooni jooksul. Seda kasutatakse harva, pigem lihtsate juhtumite puhul.

Explicit wait (nt WebDriverWait) on täpsem: see ootab konkreetset tingimust (nähtavust, klõpsatavust, olemasolu jms) ja kehtib vaid selle ühe ootuskorra kohta.

Praktikas soovitatakse kasutada ainult explicit wait'e, sest need annavad rohkem kontrolli ja väldivad ootuste kattumist, mis võib tekitada ettearvamatuid viivitusi.

2. *findElement vs findElements. Mis juhtub, kui elementi pole?*

findElement - otsib üht elementi ja tagastab selle. Kui elementi ei leita - viskab erindi "NoSuchElementException"

findElements - otsib kõiki sobivaid elemente ja tagastab loendi („list"). Kui midagi ei leita - ei viska erindit, vaid tagastab tühja loendi.

3. *NoSuchElementException vs TimeoutException. Põhjuste vahe.*

NoSuchElementException

- Tekib siis, kui driver.findElement(...) ei leia elementi üldse.
- Selenium otsib kohe DOM-ist ja kui elementi pole, viskab vea.
- Põhjus: vale locator, element pole veel loodud või eemaldati enne otsimist.

TimeoutException

- Tekib siis, kui kasutad WebDriverWait(...).until(...) ja tingimus ei täitu määratud aja jooksul.
- Näiteks: oodati, et element muutuks nähtavaks, aga see ei juhtunud enne timeout'i.
- Siin Selenium leidis või otsis korduvalt, kuid tulemus ei ilmunud õigel ajal.

4. *StaleElementReferenceException. Põhjused ja tüüplahendused.*

StaleElementReferenceException tekib siis, kui Seleniumi leitud element kaob või muutub DOM-is - ehk viide on “aegunud”.

Põhjused

- Leht laaditi uuesti või DOM uuendati (AJAX, JavaScript).
- Element eemaldati ja loodi uuesti.
- Frame või aken vahetus.

Lahendused

- Leia element uuesti enne igat tegevust.
- Kasuta explicit wait’e (WebDriverWait.until(...)).
- Oota staleness ja seejärel uuesti findElement.
- Ära kasuta @CacheLookup (see tekitab stale-vigu).

5. *CSS vs XPath. Miks 80% juhtudest CSS on eelistatud?*

- **Kiirus & mootorid**. Brauserid on aastaid optimeerinud CSS-selektori mootoreid (renderdamiseks), seega on need praktikas tavaliselt kiiremad kui XPath.
- **Lihtsus & loetavus**. CSS on lühem ja inimloetavam; XPathi võimsus tuleb hinna eest (pikk ja habras väljend). Loetavus on olulisem kui mikrosekuntiline vahe.
- **Stabiilsus.** Heade klasside/atribuutidega markupis saab sama asja CSS-iga kirjutada lühemalt ja vähem hapralt.
- **Praktiline rusikareegel**. Paljud praktikud kasutavad ID siis, kui olemas; CSS kõige muu jaoks; XPath ainult siis, kui vaja tekstipõhist või keerukat sugulust (nt ülespoole liikuda tabelites). See annabki ligikaudu jaotuse ID ~10% / CSS ~80% / XPath ~10%.

Millal siiski XPath?

Kui peab valima teksti järgi, liikuma vanema/esiaja suunas või tegema keerukaid relaativseid päringuid - siis XPath on vajalik.

6. *Page Object Model. Põhiprintsiibid.*

- **Abstraktsioon & kapseldus.** Test ei tunne HTML-i detaile; POM peidab lokatorid ja interaktsioonid meetoditesse (nt loginWith(user)), nii et UI muutus nõuab parandust ühes kohas.
- **Single Responsibility (SRP).** Üks page-objekt vastutab ühe lehe/fragmendi eest; suured lehed jaga page fragmentideks/komponentideks.
- **Puhtad meetodid, mitte utiliidid.** Eelista semantilisi tegevusi (fillShippingAddress(addr), submit()) - mitte „üldist fill(field, value)”, kui see halvendab loetavust.

- **Lokatorid privaatsed.** Hoia lokatorid page-klassis (privaatsed väljad/meetodid). Väljapoole ekspordi ainult käitumine.
- **Ilma assertideta.** Page-objekt toimib, testid kontrollivad. Assertid jäägu testikihti.
- **Navigeerimise tagastus.** Tegevus, mis viib uuele lehele, tagastab vastava page-objekti (return new CheckoutPage()), et voog oleks sujuv.
- **Ootamine keskse strateegiaga**. Kasuta explicit wait'e page-tasemel (nt BasePage), mitte Thread.sleep.
- **Ära cache'i hapraid WebElemente.** DOM muutub → leia element vahetult enne tegevust või kasuta laisk-otsingut.
- **Taaskasutus & loetavus enne kõike.** Eesmärk on vähem duplikaati, parem hooldatavus ja selged testid

7. Headless Chrome. Levinud piirangud ja vead.

Piirangud ja vead

- **Leht jääb tühjaks või valgeks** - eriti pärast Chrome'i uuendusi (nt v129).
- **Kirjatüübid ja märgid** (nt Aasia tähed) võivad kuvuda valesti, kuna headless ei laadi kõiki fonte.
- **GPU ja animatsioonid** ei tööta samamoodi nagu tavalises režiimis (pole riistvarakiirendust).
- **Ekraanitõmmised** tulevad valede mõõtudega või tühjad, kui pole seatud akna suurust.
- **Mõned veebielemendid** (nt Canvas, Shadow DOM, PDF- või meediaelemendid) ei renderdu täielikult.

Tüüplahendused

Lisa käivitamisel õiged argumendid:

```
options.add_argument("--headless")
options.add_argument("--window-size=1920,1080")
```

- Kui pilt valge: kasuta **uut headless-režiimi** (--headless=new) või lukusta Chrome'i versioon, mis töötab.
- Kui fontidega probleem: **paigalda vajalikud fondid** või lisa need konteinerisse.
- Kui GPU põhjustab vigu: lisa --disable-gpu.
- Kui midagi ei tööta üldse: kasuta **Xvfb** (virtuaalne ekraan, mis käitab Chrome'i "päris" GUI-režiimis).

8. *Miks implicit + explicit koos teevad üleliigseid viivitusi?*

Kui **implicit** ja **explicit wait** on koos, tekivad **üleliigsed viivitused**, sest:

- findElement() kasutab implicit wait'i (nt 10 s) enne, kui explicit wait hakkab tööle (nt 15 s).
- Explicit wait kutsub findElement() korduvalt - iga kord lisandub implicit ootus.
- Tulemus: tegelik ooteaeg võib olla **10 s + 15 s = kuni 20 s** või rohkem.

9. *Kuidas jätta Chrome avatuks pärast testi (detach)?*

Et Chrome jääks avatuks pärast testi, tuleb seadistada detach valik Seleniumi ChromeOptions kaudu. See valik ütleb brauserile, et ta ei sulguks pärast skripti lõppu. Funktsioon töötab ainult uuemates Seleniumi versioonides (alates 4.1.0) ja Chromiumi-põhise WebDriveriga.

10. *Java: == vs .equals() Stringide võrdlemisel. Millal true?*

== kontrollib, kas kaks muutujat viitavad samale objektile mälus, mitte kas nende sisu on sama.
.equals() võrdleb stringide tegelikku sisu.
Seega == annab true ainult siis, kui mõlemad viitavad samale stringi objektile (nt string literalid, mida Java hoiab ühiselt). .equals() annab true, kui tekst on identne, sõltumata sellest, kas objektid on erinevad.

11. *Java: List vs ArrayList. Miks väljad/argumendid tüübiga List?*

List on liides, ArrayList on selle konkreetne rakendus.
Väljade ja argumentide tüübina kasutatakse tavaliselt List, sest see lubab hiljem muuta rakendust (nt LinkedList, ArrayList) ilma koodi teisi osi muutmata. See järgib abstraktsiooni ja paindlikkuse põhimõtet.

12. *JUnit: millal assertEquals vs assertTrue(condition)?*

assertEquals kasutatakse siis, kui tahad võrrelda kahe väärtuse võrdsust ja saada automaatselt veateate, mis näitab ootust ja tegelikku tulemust.

assertTrue(condition) sobib, kui kontrollitav tingimus on juba loogiline avaldis (boolean).

Lühidalt:

- assertEquals(a, b) - kui võrdled väärtusi.
- assertTrue(a == b) -kui kontrollid loogilist tingimust.

13. WebDriverWait: visibilityOfElementLocated vs presenceOfElementLocated. Vahe.

- **presenceOfElementLocated** ootab, kuni element on olemas DOM-is, isegi kui ta pole nähtav.
- **visibilityOfElementLocated ootab**, kuni element on nii olemas kui ka nähtav (displayed ja opacity > 0).

Ehk: esimene kontrollib olemasolu, teine kontrollib nähtavust kasutaja jaoks.

14. Maven Surefire: kuidas käivitada üks testiklass või -meetod CLI-st?

Ühe testiklassi või meetodi käivitamiseks Maven Surefire'iga kasutatakse -Dtest parameetrit käsureal.

Näiteks:

- ainult üks testiklass - mvn -Dtest=KlassiNimi test
- konkreetne meetod selles klassis - mvn -Dtest=KlassiNimi#meetodiNimi test

Muster lubab ka mitut klassi või meetodit korraga (Klass1#meetod1+meetod2,Klass2).

See töötab nii JUniti kui TestNG testidega, kui Surefire-plugin on projektis õigesti seadistatud.

15. Lokaatoreid püsivamaks: millal kasutada data- atribuute ja miks?*

data-* atribuute kasutatakse siis, kui tahad teha stabiilseid ja testisõbralikke lokaatoreid, mis ei sõltu lehe kujundusest või klassinimedest.

Neid tasub lisada just testimise jaoks, näiteks data-testid="login-button", et vältida katseid, mis lagunevad iga kord, kui arendajad muudavad CSS-i või HTML-i struktuuri.

See teeb testid vähem hapraks, koodi puhtamaks ja eraldab selgelt funktsionaalse tähenduse (testi siht) visuaalsest kujundusest.